CHAPTER 8

Creating a Soil Moisture Sensor

The project in this chapter explains how to use the analog and digital ports of Intel Galileo to create a very low-cost system that measures the moisture levels in the soil for home applications. This project was demonstrated in a Maker Faire in October 2013 in Rome, Italy and the purpose was to create a project with only $4.00, excluding the cost of Intel Galileo board.

Project Details

To measure moisture in the soil, you simply measure the quantity of water in it. There are different kinds of sensors available in the market, including neutron moisture gauge sensors, frequency domain sensors, capacity sensors, and simple electrodes. These are all used in home applications and each type uses different techniques.

The electrodes are one of the most affordable solutions, and for simple applications like monitoring the plants in your home, they are good solutions.

The challenge is to create a very affordable system whereby you can use material that you usually dispose in the trash or have in your garage.

You can build your own electrode sensor using galvanized nails, such as roofing nails that are used to hold the interlocked concrete tiles.

Considering that the electrodes will be in constant contact with moisture, galvanized nails are highly recommended to avoid rusting. Rust can cause poor analog readings, forcing you to adjust the system frequently.

Each sensor consists of two electrodes, in this case, two nails separated by about two inches. The first electrode receives a voltage and the other one is connected to an analog port.

Soil has conductive properties that are affected proportionally by the amount of moisture present. If there is a good quantity of water in the soil, the electrical current propagation is good; dry soil means the propagation is bad. Of course, other factors affect soil conductivity, including the degree of salts and nutrients in the soil.

This project is very simple and its purpose is only to indicate whether the soil has enough water for the houseplants. It doesn't measure any details about soil type, water type, or mineral concentrations. This project can be applied to *any* kind of soil or water.

For practical purposes, this project will follow the same demonstration as performed at Maker Faire in Italy. The project includes two sensors, so you will be able to monitor two plants. You will have visual feedback about the sensors' readings through a group of LEDs forming a flower face, with eyes and a mouth. The mouth is used to represent emotions such as happiness and sadness, according to the level of moisture in the soil. In other words, a "happy face" will tell you if your plant has enough water; when the soil is dried out, the face will be sad. The eyes indicate which sensor is currently selected. One eye is related to sensor number 1 and the other eye relates to sensor number 2. You can switch between the sensors by pressing a push button.

You are free to change this project; for one, you could increase the number of sensors and use all six possible analog ports (A0 to A5) on the Intel Galileo instead of only two sensors.

The section called "Ideas for Improving the Project" at the end of the chapter explains how to increase the number of sensors how to use solenoid and pumps for automatic irrigation, among other ideas.

Material List

To build this project you need the components listed in Table 8-1.

Table 8-1. *Mandatory Components*

Quantity	Components
2	Green 5mm LED
8	Red 5mm LED
1	Push button
10	220 ohms 1/4 w
2	12K ohms 1/4 w
4	Galvanized nails
1	Universal board
2	Pieces of foam 3x3 inches
3 ft	Wires (at least 1/4 w)
1	Scissors

The list in Table 8-2 is not mandatory; these materials are used for aesthetic purposes during the development of the flower face, which is discussed in detail in this chapter.

Table 8-2. *Optional Materials*

Quantity	Components
1	White prime spray
1	Assorted colored papers or free paint paper samples
1	3/4 transparent tape
1	Black gel pen

Assembling the Moisture Sensors

To assemble the sensors, there are mechanical and electrical procedures you must follow. They are explained in the following sections.

Mechanical Assembly

Each sensor needs two galvanized nails and a piece of non-conductive foam in order to keep the nails separated.

For the foam, you can use the kind used to protect electronic devices packed into boxes. You might have some in your house, in the recycling, or in your office. You can also use pieces of Styrofoam.

Once you have the nails and the foam, simply insert the nails 1.5 to 2 inches apart. Figure 8-1 shows you an example of the final build.

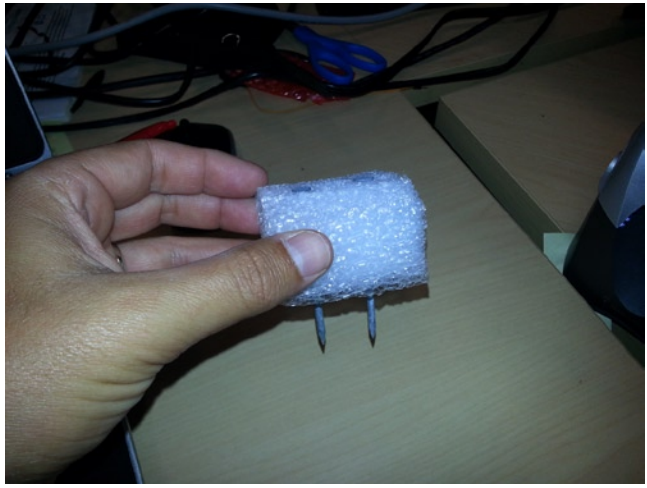

Figure 8-1. *The soil moisture sensor using nails and foam*

Once the nails are inserted into the foam, make sure they can penetrate the soil at least 1.5 inches. You will not bury the foam; you simply use it to keep the nails separated and connected to the soil.

If you do not have enough nail area at the bottom, cut the foam or use bigger nails.

Electrical Assembly

The next step is to understand how the nails are connected electrically. Therefore, remove the nails from the foam and review the image in Figure 8-2.

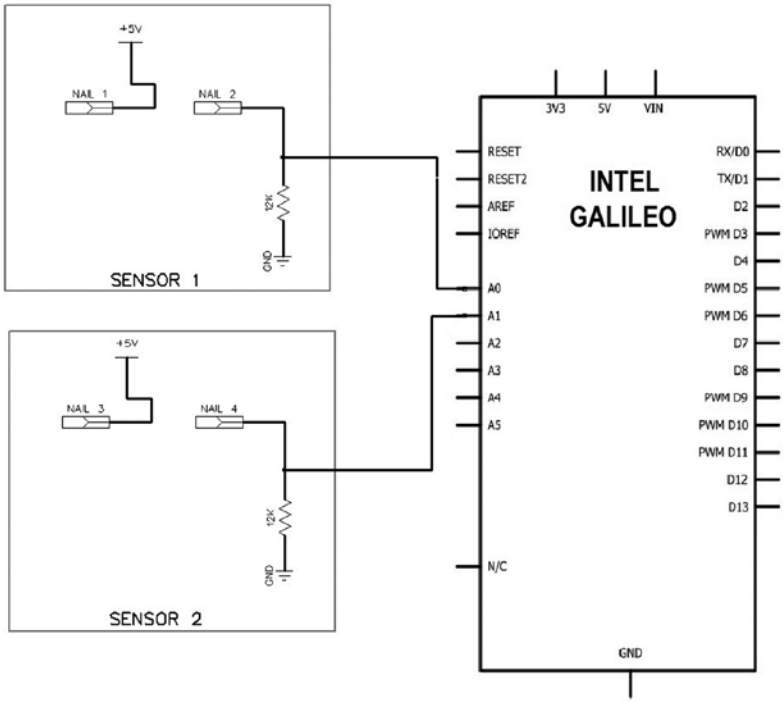

Figure 8-2. *Sensor electric connection*

As you can see in Figure 8-2, two nails are necessary to assemble one sensor. One of the nails is connected directly to the 5V port and other must be connected with the 12K Ohm resistor to an analog port. In Figure 8-2, sensor 1 is connected to A0 and sensor 2 is connected to A1.

The nails can be connected using wire wrapped around the nail's head, as demonstrated in Figures 8-3 and 8-4, or they can be soldered.

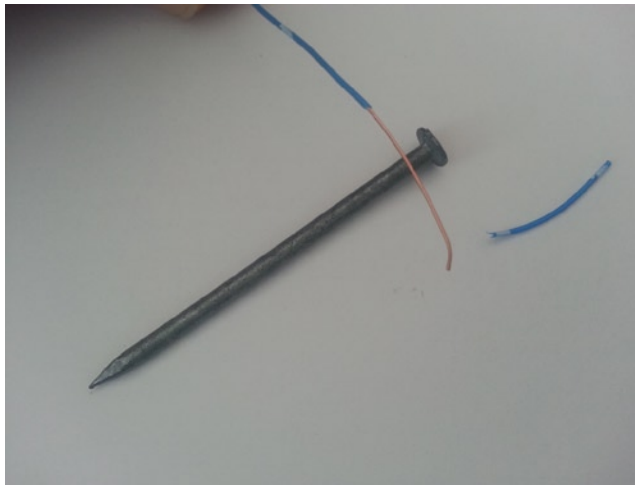

Figure 8-3. *The wire being prepared*

Figure 8-4. *Wrapping the wire around the nail's head*

Wrap the wires around each nail and place the sensors in the foam again.

The resistors work as a voltage divisor with 5V. To assemble the 5V and the resistor, connections are arranged using a universal board represented in Figure 8-2. This board will be used to assemble the flower face, which is explained later in this chapter.

Figure 8-5 shows the sensor components solder into the verse of the board (the surface without copper layer) containing a single ground (GND), the sensors cables (S1 and S2), and the common 5V connected to the sensors.

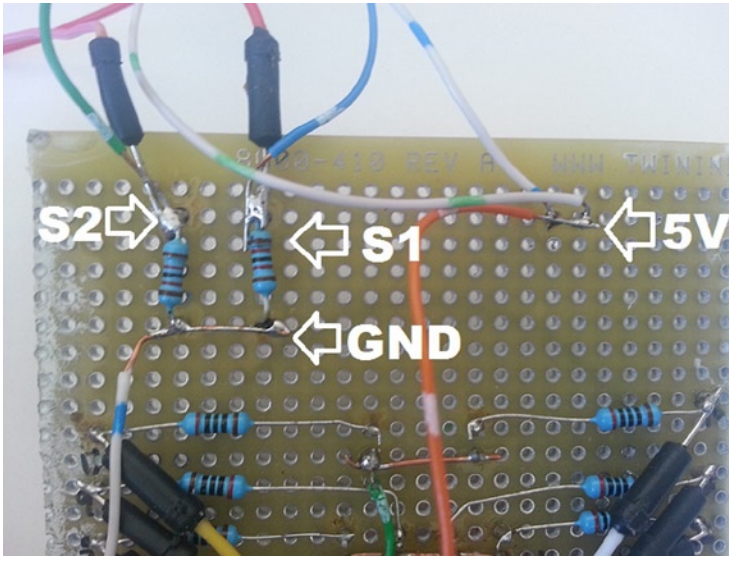

Figure 8-5. *Sensors circuits*

Using the verse of universal board and soldering the components to the backside is not normally a good approach, but in this case it is used to emulate the demo created in Italy. You will understand better in the sub-session entitled "Assembling the Flower Face" in the following pages of this chapter. You can mount your sensor circuit in a different board or create a particular PCB for each sensor, or use any other solution that you feel is appropriate for your case. For example, you can use a breadboard instead of a universal board to avoid the soldering process.

Assembling the Flower Face

The flower face will represent emotions based on how well the plants are watered. For a better user interface, the LEDs in the eyes are a different color than the LEDs used in the mouth.

Figure 8-6 shows the schematics for the flower face, where "D" means digital and "A" means analog.

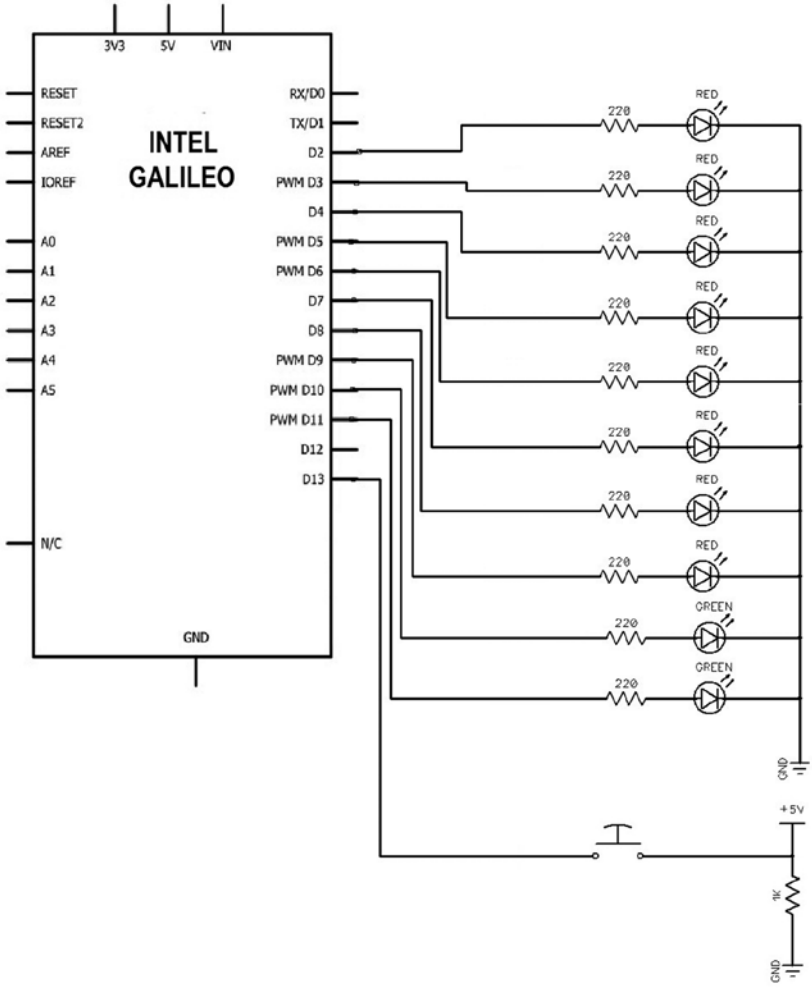

Figure 8-6. *Flower face connection*

The D10 and D11 ports are used to control the green LEDs, which inform which sensor is selected. The other ports D2 to D10 are the red LEDs and they represent the mouth. They should be aligned horizontally containing four LEDs. The push button B1 is used to select the sensor.

When this project was demonstrated at the Maker Faire in Italy, it used a universal board and the resistor was hidden in the back, as demonstrated in the Figure 8-7. It reserved space in the front of the universal board and around the LEDs in order to glue some colored paper leaves to emulate a true flower. If the resistors were in the front, it would not be possible to include these leaves. I painted the area that contains the LEDs using a white primer spray and added the circle with a black pen.

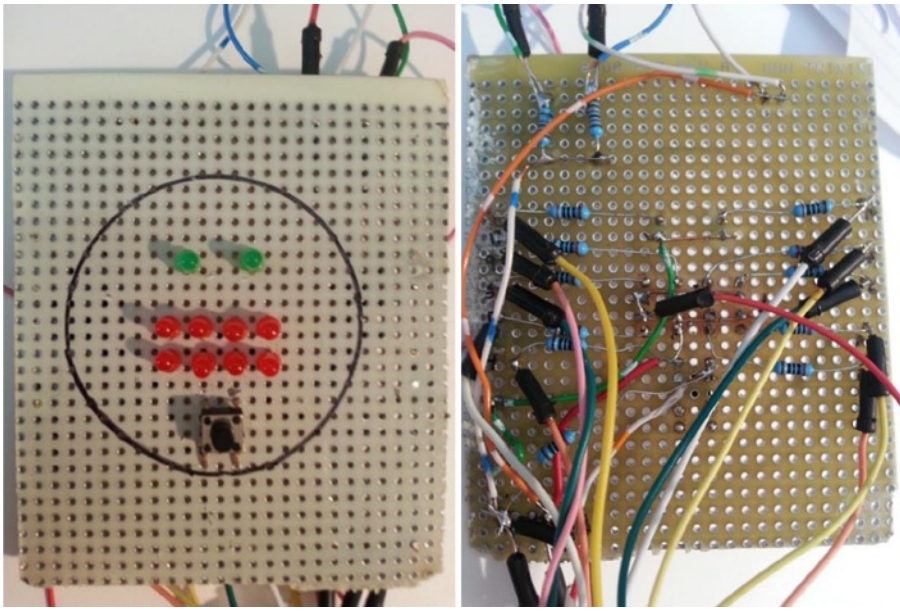

Figure 8-7. *Flower face, front (left) and back (right)*

The order of LEDs placed in the board and their respective connections with the digital ports of Intel Galileo is very important. If they aren't in proper order, the software will be a mess and will not be intuitive for other developers.

Figure 8-8 shows how each LED is arranged, with the respective digital headers in Intel Galileo boards.

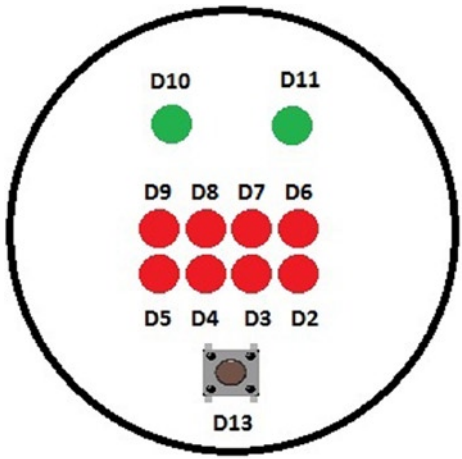

Figure 8-8. *Physical connection to Intel Galileo ports (front view)*

It is recommended that you mark each wire with the corresponding port number to avoid confusion with connections and to save time. You can do that using regular tape and paper, as shown in Figure 8-9.

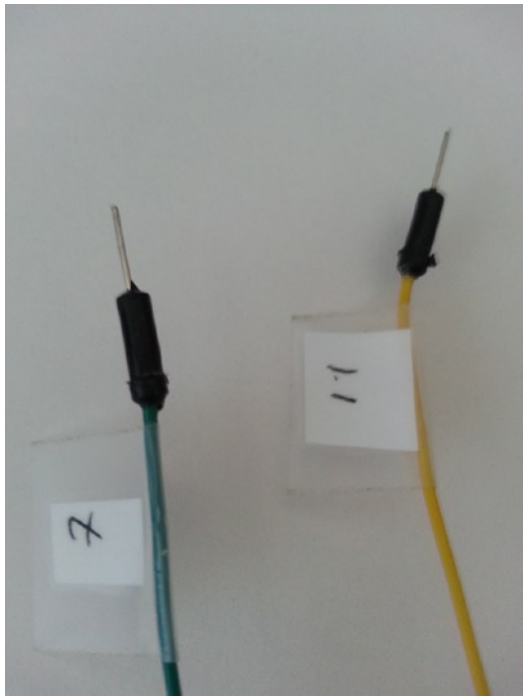

Figure 8-9. *Marking the wires with their port numbers*

Testing the Flower Face with the Software

After you have built the flower face, it is time to test if the LEDs connections are working, if they are in the right order, if the button is properly connected and switches the sensor when pressed, and adjust the button debounce. Using the Intel Galileo IDE, load the code from Listing 8-1.

Listing 8-1. flower_face_test.ino

```
// Author: Manoel Carlos Ramon
// email: manoel.c.ramon@intel.com

#define DEBUG                    0

/* Pins that define if system is ON */
#define PIN_LEFT_EYE             10
#define PIN_RIGHT_EYE            11

/* Sensor switch button */
#define PIN_SWITCH_SYSTEM        13

void clear();
int current_sensor = 0;
int button_state = 0;

int array_happy_face[2][4] = {{1, 0, 0, 1},     /* line 1 */
                              {0, 1, 1, 0}};    /* line 0 */

int array_sad_face[2][4]   = {{0, 1, 1, 0},     /* line 1 */
                              {1, 0, 0, 1}};    /* line 0 */


/*  THE MOUTH - back view
                             _____________   led 0 - pin 6
                             |  __________   led 1 = pin 7
                             |  |  _______   led 2 = pin 8
                             |  |  |  ______ led 3 = pin 9
                             |  |  |  |

                             0  0  0  0      line 1
                             0  0  0  0      line 0

                             ^  ^  ^  ^
                             |  |  |  |__  led 3 = pin 5
                             |  |  |_____ led 2 = pin 4
                             |  |________  led 1 = pin 3
                             |___________ led 0 = pin 2
*/
```

```
int lastButtonState = LOW;   // the previous reading from the input pin
long lastDebounceTime = 0;   // the last time the output pin was toggled
long debounceDelay = 500;   // adjust this value if necessary to avoid
flickering

void clear()
{
   int pin = 0;
   for (pin = 0; pin < 12; pin++)
   {
     digitalWrite(pin, LOW);
   }

}

void drawMatrix(int array[2][4])
{
  int line = 0;
  int pin = 2;
  int c = 0;
  int level = LOW;

  while (line < 2)
  {
     digitalWrite(line, LOW);

     while (c <= 3)
     {

        level = array[line][c];

        digitalWrite(pin, level);
        c++;pin++;
     }
     c=0;
     line++;
     delay(10);
  }

}

void setup() {

  if (DEBUG) Serial.begin(9600);
  // put your setup code here, to run once:
   int pin = 0;
```

```
   for (pin = 0; pin < 12; pin++)
   {
     pinMode(pin, OUTPUT);
     delay(10);
   }

   // switch button
   pinMode(PIN_SWITCH_SYSTEM, INPUT);

   // turn off all leds
   clear();

}

void checkButtonState()
{

  // read the state of the switch into a local variable:
  int reading = digitalRead(PIN_SWITCH_SYSTEM);

  // check to see if you just pressed the button
  // (i.e. the input went from LOW to HIGH),  and you've waited
  // long enough since the last press to ignore any noise:

  // If the switch changed, due to noise or pressing:
  if (reading != lastButtonState) {
    // reset the debouncing timer
    lastDebounceTime = millis();
  }

  if ((millis() - lastDebounceTime) > debounceDelay) {
    // whatever the reading is at, it's been there for longer

    // if the button state has changed:
    if (reading != button_state) {
      button_state = reading;

    }
  }

  lastButtonState = reading;

  return;
}
```

```
void loop() {

  // reading the button state
  checkButtonState();

  if( button_state != lastButtonState)
   {
      // button pressed
      current_sensor++;
      if (current_sensor > 1) current_sensor = 0;
  }

  if (DEBUG) Serial.println(current_sensor);
  if (current_sensor == 0)
  {
    //sensor 1 - only one LED on
    digitalWrite(PIN_RIGHT_EYE, HIGH);
    digitalWrite(PIN_LEFT_EYE, LOW);

    drawMatrix(array_sad_face);
  }
  else
  {
    // sensor 2 - two LEDs ON
    digitalWrite(PIN_RIGHT_EYE, HIGH);
    digitalWrite(PIN_LEFT_EYE, HIGH);

    //sad face
    drawMatrix(array_happy_face);
  }
}
```

Reviewing the Code

Before you test the circuit, review the code by checking the different functions discussed in the following sections.

setup() function

In the `setup()` function, you set the ports 2 to 9 (the mouth) and the ports defined by `PIN_LEFT_EYE` and `PIN_RIGHT_EYE` (the eyes) as output because they are the LEDs connections. `PIN_LEFT_EYE` and `PIN_RIGHT_EYE` are defined by the following code:

```
#define PIN_LEFT_EYE            10
#define PIN_RIGHT_EYE           11
```

The only port defined as input in the `setup()` function is the port that determines which button is connected. Its represented by the definition `PIN_SWITCH_SYSTEM` with port 13 by default.

```
/* Sensor switch button */
#define PIN_SWITCH_SYSTEM        13
```

loop() function

The `loop()` function checks the button state. If it is different from the previous state, that means the user pressed the button and the sensor selection must be changed.

Thus, if sensor 1 is the selected sensor, the variable `current_sensor` is zero (`0`) and a sad face will be displayed using the LEDs. Note that only one eye, represented by the green LED connected to `PIN_RIGHT_EYE`, is `ON`, which means that when there is only one eye `ON`, it's sensor 1.

Otherwise, if the user selected sensor 2, a happy face will be displayed and both "eyes," represented by the green LEDs, will be `ON`. In this case, the `current_sensor` variable value is one (`1`).

drawMatrix() function

The `drawMatrix()` function is responsible for drawing the mouth according to the matrix passed as its argument.

Each expression is defined by a double dimension matrix, represented by the `array_happy_face[][]` and `array_sad_face[][]` integer arrays.

```
int array_happy_face[2][4] = {{1, 0, 0, 1},     /* line 1 */
                              {0, 1, 1, 0}};    /* line 0 */

int array_sad_face[2][4]   = {{0, 1, 1, 0},     /* line 1 */
                              {1, 0, 0, 1}};    /* line 0 */
```

In the matrixes, the value `1` means the LED must be turned `ON` and `0` means `OFF`. Considering this, you can see that the `1`s in `array_happy_face[][]` form a smiling mouth and that they form a sad mouth in the `array_sad_face[][]`.

checkButtonState() function

The `checkButtonState()` function is responsible for determining whether the user pressed the button. For this, the `checkButtonState()` implements the same logic used by the debounce example in the IDE (see `Examples->02.Digital->Debounce`).

In the global scope, the variable called `debounceDelay` adjusts the button's debounce interval. If, during your tests, you think the button is flickering, you can increase this interval.

```
long debounceDelay = 500;   // adjust this value if necessary to avoid
flickering
```

Testing the Flower Face

This test is quite easy and fun. Just make all the connections according to the schematics. If everything is working okay, the first thing you will see is a sad face with only one eye ON, as shown in Figure 8-10.

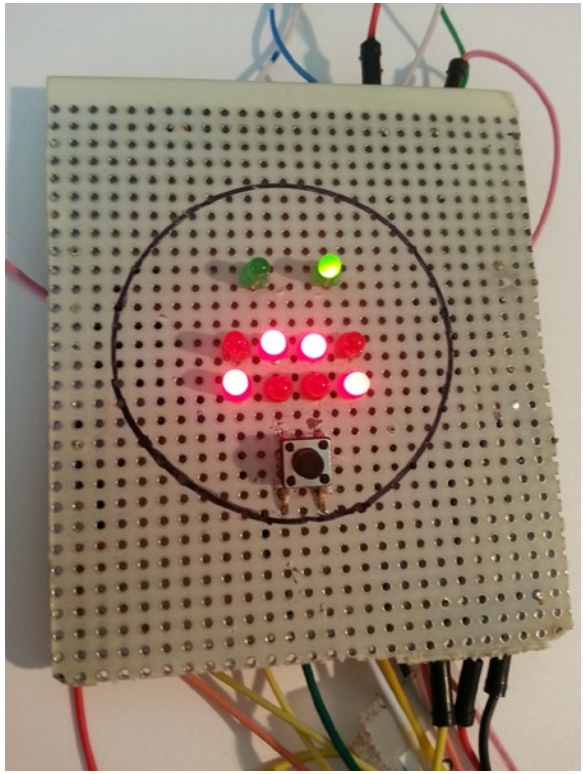

Figure 8-10. *The sad face and sensor 1 selected*

Then press the button. If the button connection and its debounce interval are working properly, you will see a happy face with two eyes ON, as shown in Figure 8-11.

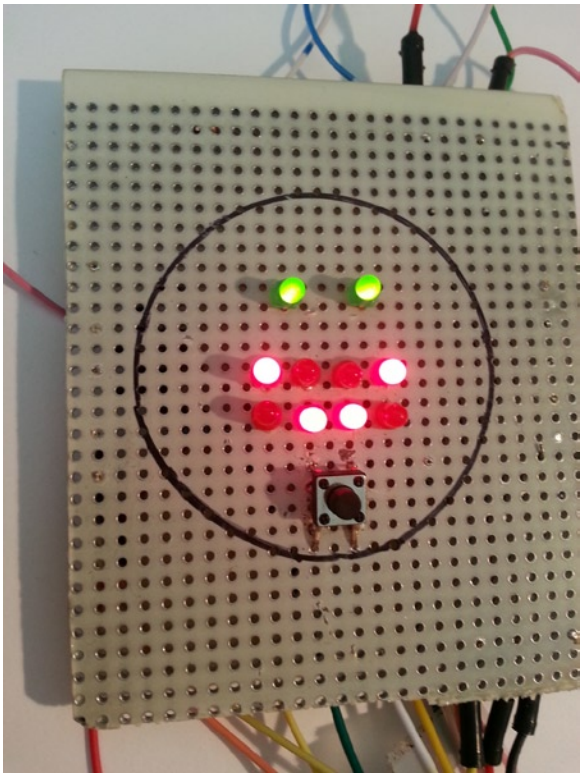

Figure 8-11. *The happy face and sensor 2 selected*

Remember, one eye `ON` means sensor 1 and two eyes `ON` means sensor 2.

Running the Project

Now it's time to run the project. You'll make a few changes to the code shown in Listing 8-1.

Calibration Procedure

If you tested the flower face and assembled the sensor correctly, the project is ready to go. You now need to integrate the logic with the sensors in the code and make some final adjustments.

The final code that joins the flower face and the sensors is called `soil_moisture.ino`. Just a few simply changes need to be made to the code.

Defining the Connections

The first change is to define where the sensors are connected and define a variable to set the initial value.

```
/* Moisture sensor - Analog Input */
#define  ANALOG_MOISTURE_SENSOR_1         A0
#define  ANALOG_MOISTURE_SENSOR_2         A1
int sensor_value = 0;
```

Setting the Boundary Values

The next change is to create a definition that will set a boundary value for when soil is wet enough. You can initiate your test using good soil; in other words, soil that contains the quantity of water that you judge good for your plants. The first thing to do is to enable to debug messages by setting the DEBUG defitition to "1".

```
#define DEBUG                    1
```

As soon as you upload the problem using the IDE, you should start the serial monitor by pressing Ctrl+Shift+M or by selecting Tools ➤ Serial-Monitor, as explained in Chapter 3. You will see a message in the serial terminal that reads "`sensor value:`" with the appropriate value. Make sure you are selecting the right sensor during this calibration.

Next, set `SOIL_ID_GOOD` to the appropriate value. In Listing 8-1, it was set to `350`, which determined if the soil is completely dried out or had enough moisture. The `350` value was tested in three different locations in the United States and in one location in Italy and worked very well for this demo. However, you should test and determine the best value for your soil and moisture level.

```
/* The analog reading boundary when soil is good */
#define  SOIL_IS_GOOD                350
```

The `loop` function is changed so that it reads the current analog port selected by the push button, compares the value, and displays the right emotion in the flower face circuit, as shown in the following excerpt from Listing 8-1.

```
void loop() {

  // reading the button state
  checkButtonState();

  if( button_state != lastButtonState)
   {
      // button pressed
      current_sensor++;
      if (current_sensor > 1) current_sensor = 0;
  }
```

```
  // reading the sensor
  switch (current_sensor)
  {
      case 0:
          sensor_value = analogRead(ANALOG_MOISTURE_SENSOR_1);

          // first sensor - one LED ON
          digitalWrite(PIN_RIGHT_EYE, HIGH);
          digitalWrite(PIN_LEFT_EYE, LOW);

      break;

      case 1:
          sensor_value = analogRead(ANALOG_MOISTURE_SENSOR_2);

          // second sensor - two LEDs ON
          digitalWrite(PIN_RIGHT_EYE, HIGH);
          digitalWrite(PIN_LEFT_EYE, HIGH);

      break;
  }

  if (DEBUG)
  {
      Serial.print("current_sensor:");
      Serial.println(current_sensor);

      Serial.print("   sensor_value:");
      Serial.println(sensor_value);
  }

  if (sensor_value >=SOIL_IS_GOOD)
  {
     drawMatrix(array_happy_face);
  }
  else
  {
     drawMatrix(array_sad_face);
  }

}
```

The code is very simple and everything is done using the digital and analog headers. In the `loop()` function will call the function `checkButtonState()`, which updates the variable `button_state`. If `button_state` is different than the previous state saved in the `lastButtonState` variable, the `current_sensor` variable changes and can assume two values—`0` or `1`. The value `0` represents the first sensor and the value `1` represents the second one.

The `switch` instruction will determine which sensor the user selected and call the `analogRead()` function, which will read the analog port that corresponds to the connected sensor and load the value to the `sensor_value` variable.

If the first sensor (case `0`) is selected, `digitalWrite()` will turn `ON` just one flower eye, because only one LED will be `HIGH`, as shown here:

```
digitalWrite(PIN_RIGHT_EYE, HIGH);
digitalWrite(PIN_LEFT_EYE, LOW
```

In case 1, two eyes will be `ON` because both LEDs will be `HIGH`:

```
digitalWrite(PIN_RIGHT_EYE, HIGH);
digitalWrite(PIN_LEFT_EYE, HIGH);
```

After this there is a debug message section, which will be displayed in the serial monitor if `DEBUG` is defined as `1`, as explained previously.

Finally, if the `sensor_value` variable reports moist soil, the `drawMatrix()` function will draw a smile in the flower face. Otherwise, a sad face is shown, as explained in the section called "Testing the Flower Face with Software" of this chapter.

Showing This Project in a Fair

When this project was used as a demo at the Maker Faire, two soil samples were used, one dried and the other one with a good quantity of moisture. The soil was in two disposable cups with sensors connected. The flower face included some colored leaves that were added using free painting paper samples you can find in construction stores like Lowes and Home Depot.

Figure 8-12 shows how the sensors were arranged in the cups and Figure 8-13 shows the flower face with the leaves when sensor 1 measured dry soil.

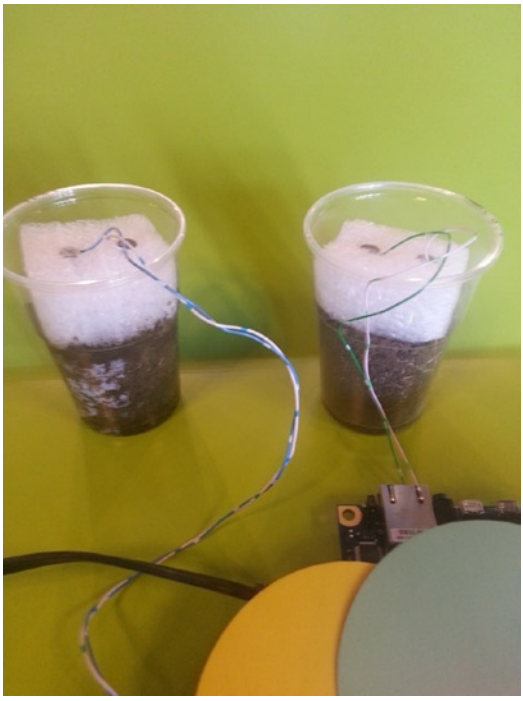

Figure 8-12. *The sensors in the cups*

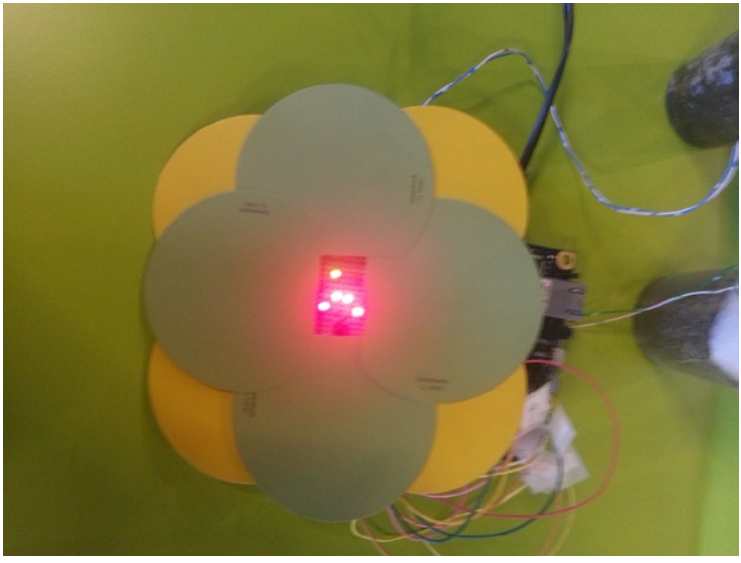

Figure 8-13. *The flower face with leaves*

Ideas for Improving the Project

This project was created knowing that it would be used as a demo in a fair or classroom. But how could you change the project to attend to your real needs? For example, sprinkler systems usually irrigate plants periodically, regardless of moisture level, which can be wasteful. The next sections discuss ideas for adjusting this project to meet real-life scenarios.

Increasing the Number of Sensors

This project uses only two sensors, but the Intel Galileo pin-out includes six analog ports. That means you can expand the number of sensors with minimal changes to the software and hardware.

If you want to have more than two sensors, instead of the flower face, you can build a simple board with multiple LEDS with each LED representing a sensor. When the LED is ON, the respective sensor indicates the plant needs water. When the LED is OFF, the soil has water enough.

Automatic Irrigation

It's possible to use this project to make an automatic irrigation system, but there are small differences when you need to irrigate a small plant or a large area.

Such differences are related to what type of device will be used in the irrigation: a pump or a solenoid valve.

When the quantity of water required is small, such as with indoor plants, you can use low voltage water pumps that operate between 3V to 9V, and you need a recepticle that collects water and distributes it to the plants. The recepticle might be a bottle, a bucket, a basin, or anything that can hold the water to be used by your plants.

When you're irrigating a large area, you'll need 12V or 24V solenoid valves that control the water that comes from a hose or pipe. These values work as a switch on and switch off. In other words, they either enables or do not enable the water flow.

In both cases, the pumps and solenoid valves work with different voltage levels and require higher levels of current than your board is able to control. Therefore, it's better to isolate the Intel Galileo and use a mechanical or solid state relay.

Figure 8-14 represents a practical circuit using a mechanical relay.

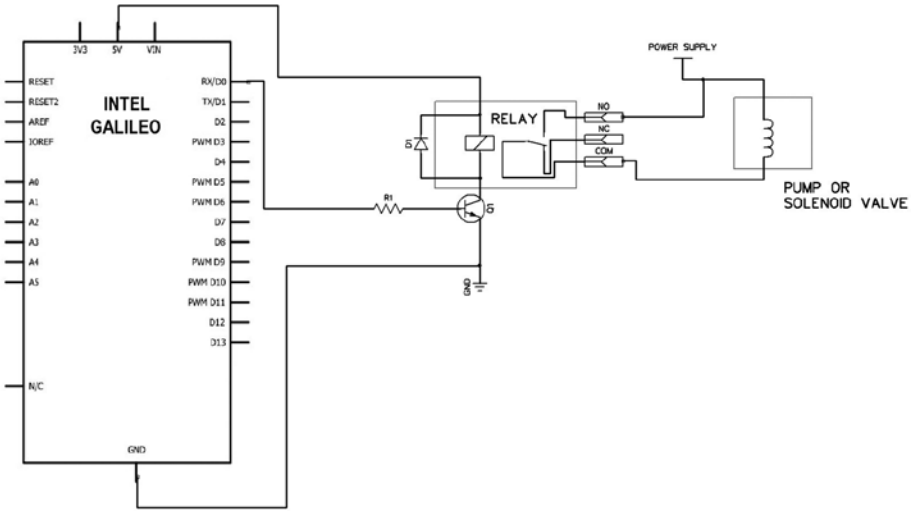

Figure 8-14. *Drive relay connected to a pump or solenoid*

The circuit shown in Figure 8-14 can be assembled using the material listed in Table 8-3.

Table 8-3. *Optional Materials*

Quantity	Components
1	R1: 1K Ohm
1	D1: 1N4001 or 1N4004
1	Q1: 2N2222A or BC548
1	Relay

There are affordable shields with relays available in the market. Figure 8-15 shows an example of a shield equipped with two relays that operates up to 30 VDC and 10A and costs less than $4.00 on eBay.

Figure 8-15. *Example of an inexpensive shield relay*

The outdoor sprinklers in landscapes are equipped with solenoid valves responsible for activating water flow. Using the circuit shown in Figure 8-14, you can manage your sprinklers by directly replacing the sprinklers' timers. Just make sure you are using the relay and power supply compatible with your sprinklers.

Using Appropriate Wires

It's possible to find wires that are more appropriate for an outdoor application at any hardware store. The demo uses simple telephony wires, but it's better to use wires that withstand moisture and water, can handle high temperatures, and work underground in the soil. Usually, they are 16 AWG and are specified as burial wires.

Using a Commercial Sensor

You can replace the galvanized nails with commercial sensors. Many electrode sensors are very affordable and are more appropriate for this application. They are usually offered with tine *potentiometers*, which allow you to make fine adjustments to the analog port reading. They can be found on eBay for less than $5.00.

Figure 8-16 shows an example of a commercial sensor that can be used with this project.

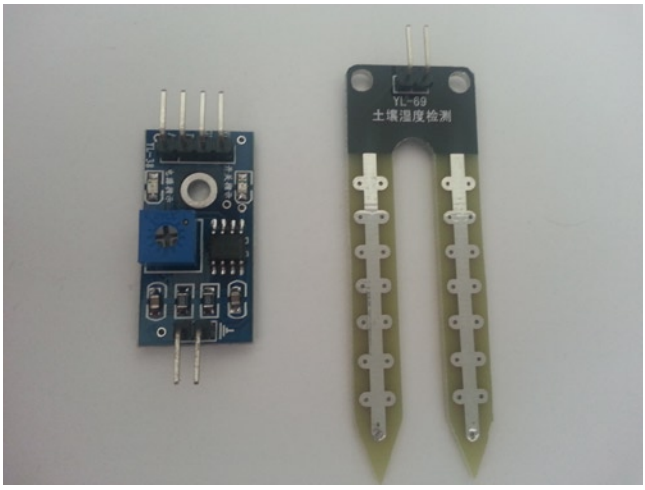

Figure 8-16. *Commercial soil sensor with potentiometer*

Tweeting

Tweeting is another way to improve this project. In the Chapter 6, you will learn how to tweet using Intel Galileo. I will talk again about this project and explain how to make the plants tweet in order to communicate when they need water.

Summary

This chapter explained how to build a project using parts that you can find in your garage, recycling, or in a construction store near you.

The next chapters offer resources that can be integrated into this project, like tweeting.